26 02 2024
Monday

* **Path with minimum effort - (Leetcode 1631)**

Given array heights of size row x columns.
Source = (0,0), destination = (row-1, col-1)
You can move left, right, up, down. find a route that requires minimum effort.

A route's effort is maximum absolute difference in heights b/w two consecutive cells of route.
Return minimum effort required for that.

I/P - heights =

	0	1	2
0	1	2	3
1	3	8	4
2	5	3	5

O/P - 1

Explanation - The route [1, 2, 3, 4, 5] has maximum abs. difference of 1 b/w consecutive cells which is minimum of all.

Code -

```
class Solution{
public:
// to check if moving left, right, up, down is safe or not
bool isSafe(int row, int col, int currX, int currY,
  int newX, int newY, vector<vector<int>>& diff)
{
  if(newX >= 0 && newY >= 0 && newX < row &&
     newY < col && diff[newX][newY] > diff[currX][currY])
   return 1;
  else
   return 0;
}

int minimumEffortPath(vector<vector<int>>& heights)
{ // create min heap
  priority_queue<pair<int,pair<int,int>>,
   vector<pair<int,pair<int,int>>>,
   greater<pair<int,pair<int,int>>>> minHeap;

  int row= heights.size();
  int col= heights[0].size();

  vector<vector<int>>diff(row,vector<int>
                          (col,INT_MAX));

  //initial state
  // set src distance = 0
   diff[0][0]=0;
  // insert entry for src position in minHeap
   minHeap.push({0,{0,0}});
```

```
// destination coordinates
int destX = row-1;
int destY = col-1;

// arrays for moving to left, right, up, down
vector<int> dx = {0,0,-1,1};
vector<int> dy = {-1,1,0,0};

while(!minHeap.empty()){
  //fetch inserted info. from minHeap
    auto topPair = minHeap.top();
    minHeap.pop();
    int currDiff = topPair.first;
    auto currNodeIndex = topPair.second;
    int currX = currNodeIndex.first;
    int currY = currNodeIndex.second;

  //now, move to all 4 directions
    for(int i=0; i<4; i++){
      int newX = currX + dx[i];
      int newY = currY + dy[i];

    //if going to new position is safe, then move
      if(isSafe(row,col,currX,currY,newX,newY,diff))
      {
        //calculate maximum abs. difference
        int absDiff = abs(heights[currX][currY]-heights[newX]
                                                  [newY]);

        int maxDiff = max(currDiff, absDiff);
```

```
diff[newX][newY] = min(maxDiff, diff[newX][newY]);

//if position is not destination, create new entry
//in minHeap
if(newX != destX || newY != destY){
    minHeap.push({diff[newX][newY], {newX, newY}});
  }
 }
}
}
return diff[row-1][col-1];
}
};
```

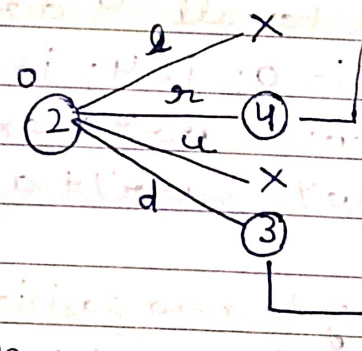
Let heights =

	0	1
0	2	4
1	3	7

diff =

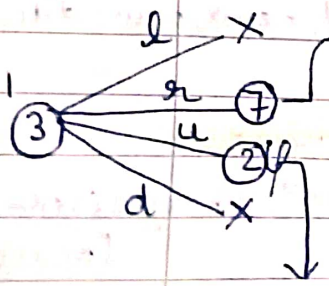
	0	1
0	0	∞ 2
1	∞ 1	∞ 43

② (1,3)
③ (2,4)
① (0,2)



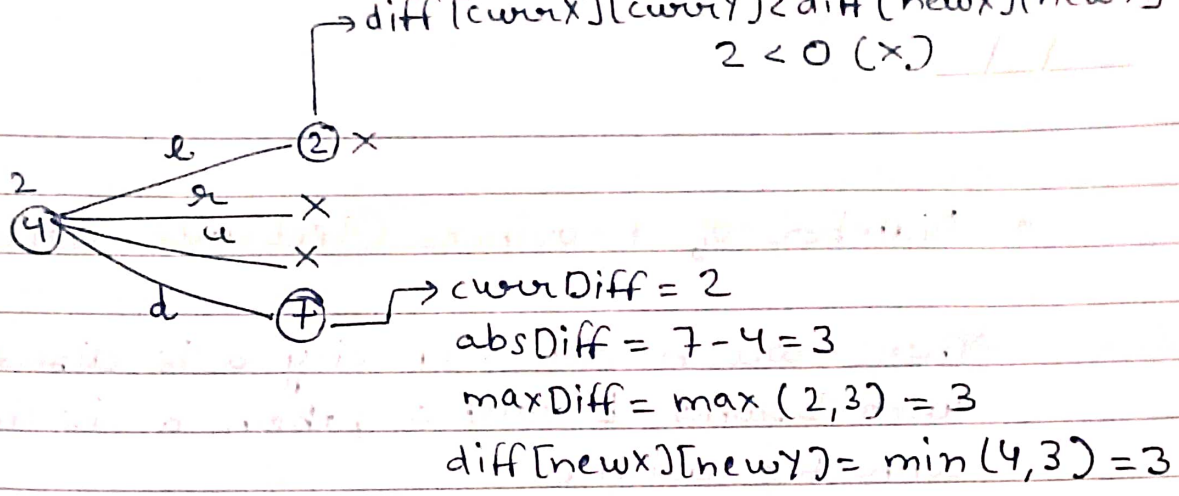
currDiff = 0
absDiff = 2-2 = 2
maxDiff = max(0,2) = 2
diff[newX][newY] = min(2,∞)

currDiff = 0
absDiff = 3-2 = 1
maxDiff = max(0,1) = 1
diff[newX][newY] = min(1,∞)



currDiff = 1
absDiff = 7-3 = 4
maxDiff = max(1,4) = 4
diff[newX][newY] = min(4,∞)

diff[currX][currY] < diff[newX][newY]
1 < 0 (X)



So, diff[row-1][col-1] = diff[1][1] = 3